

## **consultorioABM**

- I. Introducción
- II. Funcionalidades
- III. Instrucciones
- IV. Recursos Utilizados
- V. Descripción del código

## **I. Introducción**

consultorioABM es el resultado de un proyecto pensado para demostrar de forma práctica los conocimientos adquiridos durante mi formación como desarrollador full Stack. Mi motivación para el desarrollo de esta aplicación fue la idea de poder crear algo que me permitiera integrar de forma práctica todos los conocimientos que fui adquiriendo en este período de formación.

Para poder lograr esto, se me ocurrió pensar una aplicación con funcionalidades CRUD o ABM que permita al usuario crear, leer, actualizar y eliminar datos.

## **II. Funcionalidades**

Esta pequeña aplicación fue pensada con el propósito de llevar un registro de los pacientes que se van atendiendo dentro un consultorio multidisciplinario, además permite llevar el registro del departamento al cual pertenece dicho paciente, facilitando el seguimiento de los tratamientos que realizan el paciente en torno a las diferentes disciplinas dentro del mismo consultorio.

Para este propósito, consultorioABM cuenta con la capacidad de visualizar una lista de los pacientes actuales del consultorio, esta lista es facilitada por una base de datos de tipo relacional.

Además, cuenta con la capacidad de modificar los datos personales de los pacientes y con la capacidad tanto de agregar nuevos pacientes a la lista como de eliminarlos.

Para poder modificar a los pacientes registrados y agregar nuevos cuenta con la capacidad de modificar o agregar nuevos departamentos o especialidades para el consultorio. De forma que la lista de pacientes siempre se estará sincronizada con la lista de departamentos.

### III. Instrucciones

Luego de compilar la Api y el proyecto en Angular, acceder a la dirección <http://localhost:4200/>, o agregar la opción --open al ejecutar ng serve en Angular.











Al ingresar a la dirección web de la aplicación por defecto se abrirá la pestaña de Inicio en donde se encuentra una breve introducción a la aplicación y las instrucciones para acceder a sus funcionalidades.

#### 1) Acceder al registro de pacientes o departamentos

Para esto es necesario hacer click en la sección apropiada de la barra de navegación.













Esta acción va a desplegar la lista de los pacientes o departamentos registrados de forma automática.

#	Nombre	Apellido	Departamento	Opciones
1	Carlos	Perez	Psicología	 
2	Martín	Rodriguez	Psiquiatría	 
3	Marcelo	Fernandez	Odontología	 
4	Diego	Fernandez	Clínica Médica	 
5	Martin	Lescano	Psicopedagogía	 

#### 2) Agregar un nuevo elemento

Para agregar un nuevo elemento a las listas, dirigirse a la sección apropiada, en la parte derecha de la tabla se encontrará el botón denominado “Agregar paciente/departamento”. Se debe hacer click ese botón para desplegar el formulario para agregar el nuevo elemento de la lista.

#	Nombre	Apellido	Departamento	Opciones
1	Carlos	Perez	Psicología	 
2	Martín	Rodriguez	Psiquiatría	 
3	Marcelo	Fernandez	Odontología	 
4	Diego	Fernandez	Clínica Médica	 
5	Martin	Lescano	Psicopedagogía	 

Nombre

Apellido

Departamento


Luego de completar el formulario con los datos requeridos es necesario hacer click en el botón “Agregar” para confirmar la operación.

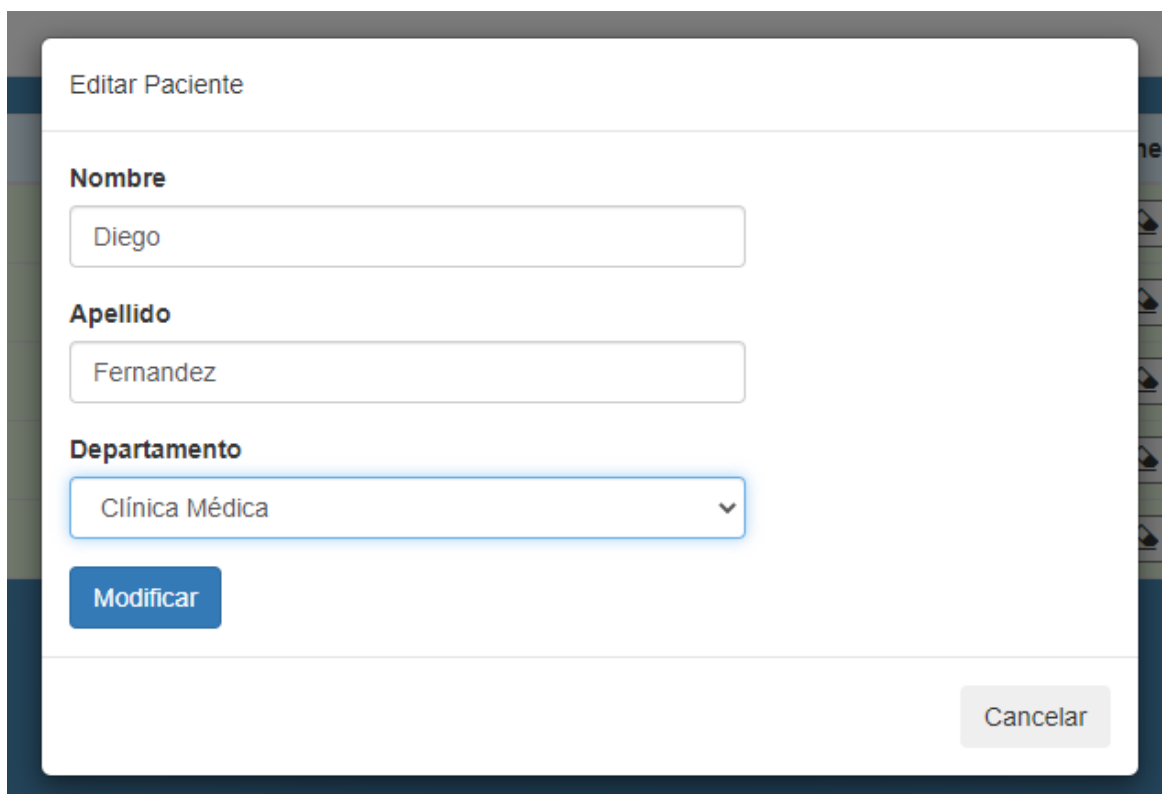
\* En el caso de los pacientes, dentro de la sección departamentos del formulario para agregar un nuevo paciente figuran los diferentes departamentos que existen disponibles, esta lista se actualiza de forma automática cuando se añade un nuevo departamento.

\*\* En el caso de querer agregar un departamento, solo es necesario aclarar su nombre y luego hacer click en el botón “Agregar”.

\*\*\* Si la operación falla puede deberse a un bug, por lo que es recomendable volver a intentar.


### 3) Modificar datos de un elemento

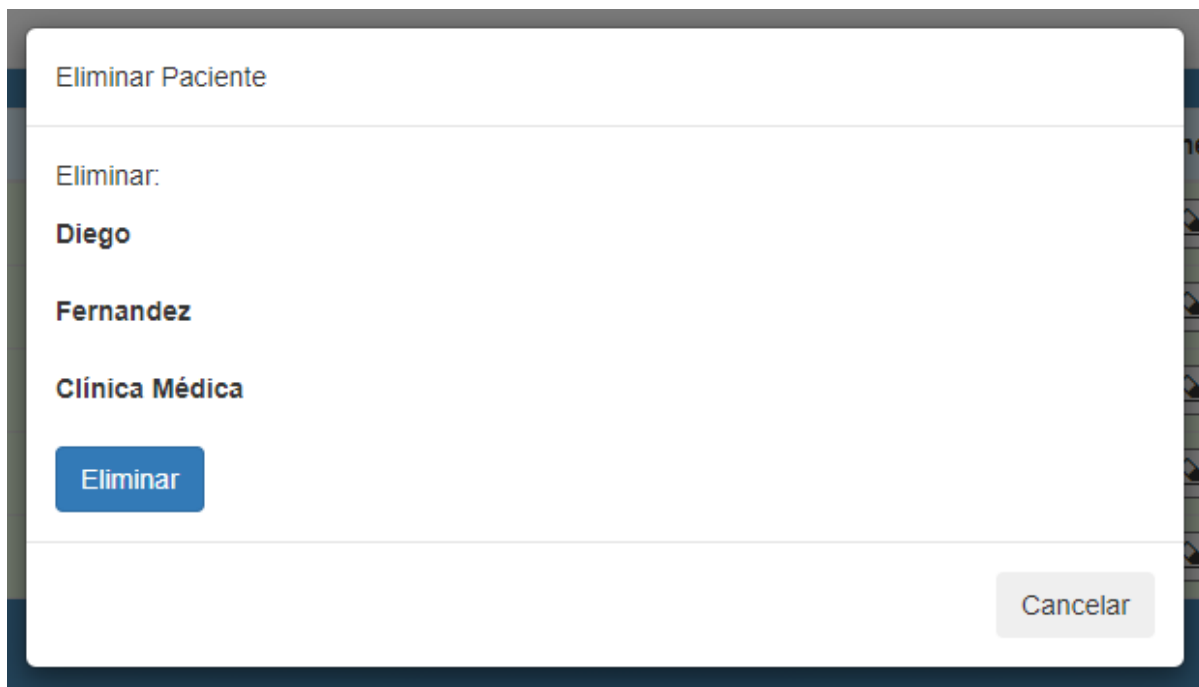
Ya sea un paciente, o un departamento, es posible modificar sus datos haciendo click en el botón  dentro de la columna “Opciones”. Esto desplegará el formulario con los datos a modificar. Luego de realizar las correcciones necesarias se debe hacer click en el botón “Modificar”.



El formulario, titulado "Editar Paciente", contiene tres campos de entrada: "Nombre" con el valor "Diego", "Apellido" con el valor "Fernandez", y "Departamento" con un menú desplegable que muestra "Clínica Médica". Debajo de estos campos hay un botón azul "Modificar". En la esquina inferior derecha del formulario hay un botón gris "Cancelar".

### 4) Eliminar un elemento

Para esto es necesario hacer click en el botón , perteneciente al elemento que se desea eliminar, que se encuentra en la columna “Opciones”. Luego de esto se abrirá una ventana en donde se deben confirmar el elemento a eliminar, luego de estar seguro, hacer click en “Eliminar”.



#### IV. Recursos Utilizados

##### Back end:

- **SQLServer / SQL Server Management Studio:** para la creación y administración de la base de datos denominada "ProyectoBD". La cual contiene las tablas con la info de los pacientes y los departamentos.
- **.NET Framework - Entity Framework - Visual Studio:** para el desarrollo de la REST-API denominada "consultorioABM". La cual permite la comunicación entre dos aplicaciones, en este caso va a permitir la comunicación entre el entorno visual (front end) y la base de datos (back end) por medio de servicios específicos.

Para lograr esto se utilizó el entorno de desarrollo integrado (IDE) Visual Studio junto con el lenguaje de programación C# y el método de mapeo objeto-relacional (ORM) denominado Entity Framework, ya que este posibilita la capacidad de acceder y manipular datos que se encuentren dentro de una base de datos como si fueran objetos, permitiendo realizar un conjunto amplio de operaciones de forma eficiente y productiva.

##### Front end:

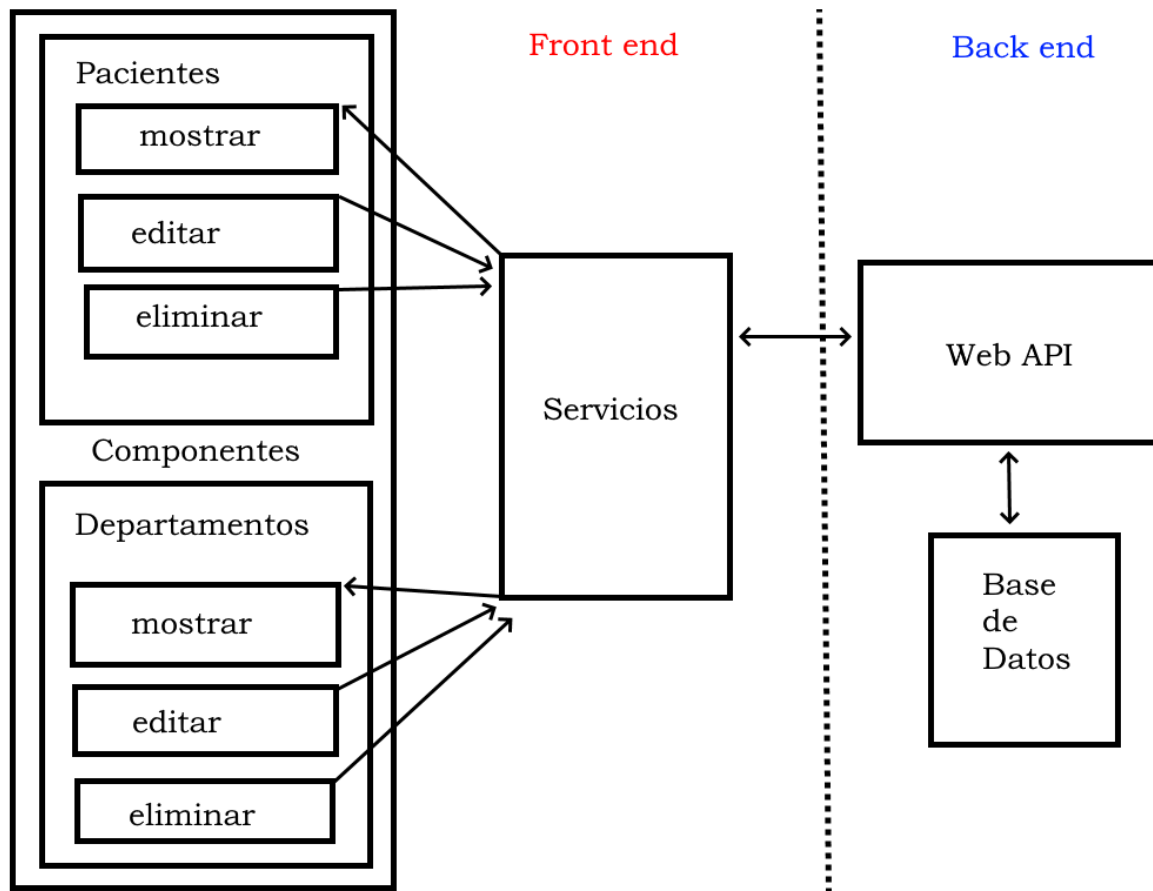
- **HTML5:** lenguaje de marcado para la elaboración de paginas web. Define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Permite indicar la estructura de nuestro documento mediante etiquetas.
- **CSS3:** lenguaje de diseño grafico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para

establecer el diseño visual de los documentos web e interfaces de usuario escritas.

- **JavaScript:** es un lenguaje de programación interpretado, definido como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente para permitir mejoras en la interfaz de usuario y páginas web dinámicas.
- **Bootstrap3:** es un conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.
- **Angular2+:** es un framework para aplicaciones web desarrollado en TypeScript, de código abierto, que se utiliza para crear y mantener aplicaciones web de página-única (da una experiencia más fluida al usuario, como si fuera una aplicación de escritorio, ya que todos los códigos se cargan una sola vez, o los recursos necesarios se cargan de forma dinámica cuando lo requiera la página, como respuesta a las acciones del usuario). Una de las mayores ventajas de angular es la utilización de componentes, los cuales son una porción de código que pueden utilizarse y reutilizarse a discreción dentro del entorno y entre otras aplicaciones en otros proyectos hechos con Angular.

## V. Descripción del código

A la hora de diseñar este proyecto, empecé por armar el siguiente esquema. El cual incluye los componentes principales y los subcomponentes que quería utilizar en Angular, así como una breve idea de como debería ser el flujo de los datos para que la aplicación cumpla su función.



### ▪ Back end: Web API y Base de datos.

Para el desarrollo de la API se implementó un abordaje basado en Entity Framework, luego de crear la base de datos, sus tablas y contenidos, esto permitió generar el código necesario para representar a la base de datos en la API mediante clases, así como desarrollar los métodos necesarios para la comunicación con la misma.

Modelos:

Los modelos generados son la representación de las tablas de la base de datos, en este caso contamos con dos modelos: Departamentos y Pacientes. Dichos modelos incluyen como propiedades las columnas correspondientes a cada tabla.

10 references

```
public partial class Departamentos
```

```
{
```

```
    [Key]
```

3 references

```
    public int IdDepartamento { get; set; }
```

```
    [StringLength(50)]
```

1 reference

```
    public string NombreDepartamento { get; set; }
```

```
}
```

12 references

```
public partial class Pacientes
```

```
{
```

```
    [Key]
```

3 references

```
    public int IdPaciente { get; set; }
```

```
    [StringLength(50)]
```

1 reference

```
    public string NombrePaciente { get; set; }
```

```
    [StringLength(50)]
```

1 reference

```
    public string ApellidoPaciente { get; set; }
```

```
    [StringLength(50)]
```

1 reference

```
    public string Departamento { get; set; }
```

```
    [Column(TypeName = "date")]
```

0 references

```
    public DateTime? InicioTratamiento { get; set; }
```

```
}
```

Controladores:

Los controladores incluyen los métodos necesarios para el intercambio de información entre la web API y la base de datos. Que es lo que permitirá que nuestra aplicación lleve a cabo todas sus funciones.

Los controladores para pacientes y para departamentos son similares. Ejemplos de controladores:



0 references

```
public class DepartamentosController : ApiController
```

```
{  
    private Model1 db = new Model1();
```

```
    // GET: api/Departamentos
```

0 references

```
    public IQueryable<Departamentos> GetDepartamentos()  
    {  
        return db.Departamentos;  
    }
```

```
    // GET: api/Departamentos/5
```

```
    [ResponseType(typeof(Departamentos))]
```

0 references

```
    public IHttpActionResult GetDepartamentos(int id)  
    {  
        Departamentos departamentos = db.Departamentos.Find(id);  
        if (departamentos == null)  
        {  
            return NotFound();  
        }  
  
        return Ok(departamentos);  
    }
```

```
    // PUT: api/Departamentos/5
```

```
    [ResponseType(typeof(void))]
```

0 references

```
    public IHttpActionResult PutDepartamentos(int id, Departamentos departamentos)  
    {  
        if (!ModelState.IsValid)  
        {  
            return BadRequest(ModelState);  
        }  
  
        if (id != departamentos.IdDepartamento)  
        {  
            return BadRequest();  
        }  
  
        db.Entry(departamentos).State = EntityState.Modified;  
  
        try  
        {  
            db.SaveChanges();  
        }  
        catch (DbUpdateConcurrencyException)  
        {  
            if (!DepartamentosExists(id))  
            {  
                return NotFound();  
            }  
            else  
            {  
                throw;  
            }  
        }  
  
        return StatusCode(HttpStatusCode.NoContent);  
    }  
}
```

```

// POST: api/Departamentos
[ResponseType(typeof(Departamentos))]
0 references
public IActionResult PostDepartamentos(Departamentos departamentos)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    db.Departamentos.Add(departamentos);
    db.SaveChanges();

    return CreatedAtRoute("DefaultApi", new { id = departamentos.IdDepartamento }, departamentos);
}

// DELETE: api/Departamentos/5
[ResponseType(typeof(Departamentos))]
0 references
public IActionResult DeleteDepartamentos(int id)
{
    Departamentos departamentos = db.Departamentos.Find(id);
    if (departamentos == null)
    {
        return NotFound();
    }

    db.Departamentos.Remove(departamentos);
    db.SaveChanges();

    return Ok(departamentos);
}

```

Dichos controladores permiten que realizar las funciones básicas que queremos alcanzar: leer, crear, modificar y eliminar datos.

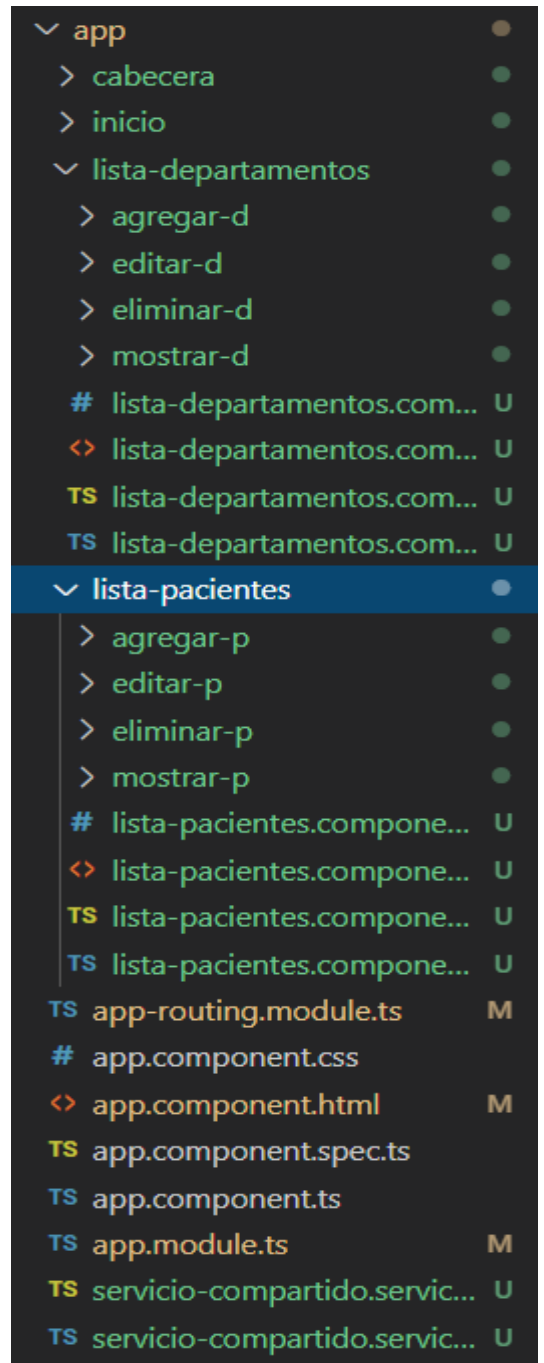
- Front end: Angular, componentes y servicios.

Para el desarrollo del front end decidí utilizar el framework Angular, ya que es la herramienta que aprendí a utilizar durante mi formación y además permite el desarrollo de aplicaciones de forma eficiente y organizada.

Para empezar, decidí crear los componentes, subcomponentes y servicios necesarios que iba a necesitar para la aplicación completa e instalar bootstrap3 en el proyecto con angular CLI, para luego comenzar a escribir el código para cada componente o servicio.

Luego de esto, la estructura del proyecto quedó de esta manera.

Organizar los componentes de esta forma me permitió insertar a cada uno en su posición definitiva dentro de la página web para luego poder ir escribiendo el código de cada uno de forma separada y organizada.



## Servicios

Los servicios permiten realizar las consultas Http necesarias para comunicarse con la API para realizar las funciones necesarias. Cada componente principal cuenta con 4 servicios, que corresponden a las diferentes operaciones que se pueden realizar, GET (obtener la lista de pacientes o departamentos), POST (agregar un paciente o departamento), PUT (modificar un paciente o departamento) y DELETE (eliminar un paciente o departamento).

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ServicioCompartidoService {

  readonly APIUrl="https://localhost:44369/api";

  constructor(private http: HttpClient) { }

  // Servicios API Pacientes
  obtenerPacientes(): Observable<any[]> {
    return this.http.get<any>(this.APIUrl + '/pacientes');
  }

  crearPaciente(val: any){
    return this.http.post(this.APIUrl + '/pacientes', val);
  }

  modificarPaciente(id: any, val: any){
    return this.http.put(this.APIUrl + '/pacientes/' + id, val)
  }

  eliminarPaciente(val:any){
    return this.http.delete(this.APIUrl + '/pacientes/' + val)
  }
}

```

## Routing

El routing de la aplicación fue realizado de la siguiente forma, dentro del archivo app-routing.module.ts.

```

import { InicioComponent } from './inicio/inicio.component';
import { ListaPacientesComponent } from './lista-pacientes/lista-pacientes.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ListaDepartamentosComponent } from './lista-departamentos/lista-departamentos.component';

const routes: Routes = [
  {path: 'pacientes', component: ListaPacientesComponent},
  {path: 'departamentos', component: ListaDepartamentosComponent},
  {path: '', component: InicioComponent},
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

## Componentes

Los componentes principales de la aplicación son los siguientes:  
cabecera - inicio - lista-departamentos - lista-pacientes.

## ▪ Cabecera

En este componente se aloja la barra de navegación que permite desplazarse por las diferentes secciones de la aplicación. Su contenido es el siguiente:

```
<!-- Logo -->
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a href="#" class="navbar-brand">
        consultorioABM
      </a>
    </div>
    <!-- Barra de navegación -->
    <div class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li><a href="http://localhost:4200">Inicio</a></li>
        <li><a href="http://localhost:4200/pacientes">Pacientes</a></li>
        <li><a href="http://localhost:4200/departamentos">Departamentos</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Se encuentra insertado de forma global en el componente principal de Angular, app.component.html. En cambio, los demás componentes se utilizan mediante la directiva router-outlet de Angular.

## ▪ Inicio

En el componente de Inicio se encuentra la dirección que se abre por defecto al ingresar a la url de la aplicación. Contiene el título de la aplicación, el cual se inserta mediante string interpolation, y una breve descripción e instrucciones sobre la aplicación.

```
<div class="container-fluid">
  <div class="row">
    <h1>
      {{ titulo }}
    </h1>
    <hr>
    <!-- Descripción -->
    <h4>
      {{ titulo }} es un proyecto
      pensado y llevado a cabo en Angular, .Net y SQLServer.
    </h4>
    <p>
      La intención de este proyecto es poder demostrar de forma práctica los conocimientos adquiridos durante
      mi formación como desarrollador Full Stack. Mi intención es la de poder crear una aplicación que me
      permita integrar
      de forma práctica las cosas que fui aprendiendo durante este período.
    </p>
  </div>
</div>
```

## ▪ Lista-pacientes

Este componente cuenta con disposición y funcionalidades similares a las del componente lista-departamentos, por lo que se utilizará para ejemplificar a ambos.

El componente principal lista-pacientes obra como contenedor para los sub-componentes que se encargan de mostrar la tabla de pacientes y que permiten agregar más pacientes. Para administrar la ejecución de los subcomponentes se utilizan las funcionalidades provistas por Angular conocidas como event binding y

directivas, en este caso se utiliza la directiva `ngIf` para ejecutar el componente que permite agregar pacientes de forma dinámica.

```
<div class="container divApp">
  <div class="row">
    <div class="col-lg-10">
      <app-mostrar-p></app-mostrar-p>
    </div>
    <div class="col-lg-2">
      <!-- cambia el estado de formAgregarPaciente a true -->
      <button class="btn btn-success" (click)="agregarP()">
        Agregar Paciente
      </button>
    </div>
  </div>
  <hr>
  <!-- cambia el estado de formAgregarPaciente a false -->
  <div *ngIf="formAgregarPaciente" class="col-lg-2">
    <button class="btn btn-danger" (click)="cancelarAgregarP()">
      Cancelar
    </button>
  </div>
</div>
<div class="row">
  <!-- cuando formAgregarPaciente es true activa el componente para agregar pacientes -->
  <div *ngIf="formAgregarPaciente" class="col-lg-12">
    <app-agregar-p></app-agregar-p>
  </div>
</div>
</div>
```

```
@Component({
  selector: 'app-lista-pacientes',
  templateUrl: './lista-pacientes.component.html',
  styleUrls: ['./lista-pacientes.component.css']
})
export class ListaPacientesComponent implements OnInit {

  paciente: any;
  formAgregarPaciente: boolean = false;
  constructor() { }

  ngOnInit(): void {
  }

  agregarP(){
    this.formAgregarPaciente = true;
  }

  cancelarAgregarP(){
    this.formAgregarPaciente = false;
  }
}
```

- Mostrar pacientes

Este subcomponente despliega la tabla en donde se visualizan los datos de los pacientes que provienen desde la base de datos por medio de nuestra web API. La

directiva ngFor se utilizó en este caso para rellenar los campos de la tabla de forma dinámica.

```
<table class="table table-bordered table-hover">
  <thead>
    <tr class="info">
      <th th scope="col">#</th>
      <th th scope="col">Nombre</th>
      <th th scope="col">Apellido</th>
      <th th scope="col">Departamento</th>
      <th th scope="col">Opciones</th>
    </tr>
  </thead>
  <tbody>
    <!-- ciclo que recorre los elementos del vector listaP, i toma el valor de los indices para la tabla -->
    <tr *ngFor="let paciente of listaP, let i = index" class="success">
      <td>
        {{ i + 1 }}
      </td>
      <td>
        {{ [paciente.NombrePaciente] }}
      </td>
      <td>
        {{ paciente.ApellidoPaciente }}
      </td>
      <td>
        {{ paciente.Departamento }}
      </td>
      <td class="botones">
        <!-- boton editar/eliminar -->
        <button type="button" data-toggle="modal" data-target="#editModal" (click)="editClick(paciente)">
          <span class="glyphicon glyphicon-edit"> </span>
        </button>
        <button type="button" data-toggle="modal" data-target="#editModal" (click)="elimClick(paciente)">
          <span class="glyphicon glyphicon-erase"> </span>
        </button>
      </td>
    </tr>
  </tbody>
</table>
```

Este subcomponente también cuenta con los botones necesarios para ejecutar los subcomponentes que permiten modificar o eliminar datos de la tabla, además cuenta con los eventos y la estructura del modal que se despliega al hacer click en dichos botones.

```
<!-- Modal editar paciente -->
<div class="modal fade" id="editModal" tabindex="-1" role="dialog" aria-labelledby="editModal" aria-hidden="true"
data-backdrop="static" data-keyboard="false">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="editModal"> {{ tituloModal }} </h5>
      </div>
      <div class="modal-body">
        <app-editar-p
          [pac]="pac" *ngIf="activarEditar"></app-editar-p>
        <app-eliminar-p
          *ngIf="activarEliminar" [pacId]="pacId" ></app-eliminar-p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-dismiss="modal" (click)="cancelarClick()">Cancelar</button>
      </div>
    </div>
  </div>
</div>
```

Dentro del archivo .ts de dicho componente se encuentra la lógica necesaria para ejecutar todas sus funciones, como por ejemplo: aquí se utiliza el servicio obtenerPacientes() para obtener la información de los pacientes que se encuentra en la base de datos y rellenar la tabla con esa información.

```

export class MostrarPComponent implements OnInit {

  constructor(private servicio: ServicioCompartidoService) { }

  listaP: any = [];
  activarEditar: boolean = false;
  activarEliminar: boolean = false;
  pac: any;
  pacId: any;
  tituloModal: any;

  ngOnInit(): void {
    this.mostrarPacientes();
  }

  mostrarPacientes(){
    this.servicio.obtenerPacientes().subscribe(datos => {
      this.listaP = datos;
    });
  }

  editClick(val: any){
    this.activarEditar = true;
    this.pac=val;
    this.tituloModal = "Editar Paciente";
  }

  elimClick(val: any){
    this.activarEliminar = true;
    this.pacId = val;
    this.tituloModal = "Eliminar Paciente";
  }

  cancelarClick(){
    location.reload();
  }
}

```

mostrar-p.component.ts

#### ▪ Agregar pacientes

Este subcomponente se encuentra insertado dentro del componente principal lista-pacientes, al hacer click en el botón “Agregar paciente” se despliega un formulario de forma dinámica mediante la directiva ngIf, este formulario permite proveer los datos necesarios para crear un nuevo paciente y visualizarlo dentro de la tabla.

Para realizar esta acción se utiliza el abordaje de formularios provisto por Angular, conocido como “reactive driven” en el cual se especifica la estructura del formulario en forma de objeto de JavaScript, en combinación con el servicio creado anteriormente para realizar acciones post, en este caso denominado como crearPaciente().



```

<!-- Formulario para agregar pacientes -->
<div class="container">
  <div class="row">
    <div class="col-xs-12 col-sm-10 col-md-8">
      <form [formGroup]="pacienteForm" (ngSubmit)="onSubmit(pacienteForm.value)">
        <div class="form-group">
          <label for="NombrePaciente">Nombre</label>
          <input type="text" id="NombrePaciente" class="form-control"
            formControlName="NombrePaciente">
        </div>
        <div class="form-group">
          <label for="ApellidoPaciente">Apellido</label>
          <input type="text" id="ApellidoPaciente" class="form-control"
            formControlName="ApellidoPaciente">
        </div>
        <div class="form-group">
          <label for="Departamento">Departamento </label>
          <select name="Departamento" id="Departamento" class="form-control"
            formControlName="Departamento">
            <option *ngFor="let dep of listaD" > {{ dep.NombreDepartamento }} </option>
          </select>
        </div>
        <hr>
        <button class="btn btn-primary" type="submit">
          Agregar
        </button>
      </form>
    </div>
  </div>
</div>

```

```

export class AgregarPComponent implements OnInit {

  listaD: any = [];
  pacienteForm!: FormGroup;

  constructor(private servicio: ServicioCompartidoService) {}

  ngOnInit(): void {
    this.pacienteForm = new FormGroup([
      'NombrePaciente': new FormControl(null),
      'ApellidoPaciente': new FormControl(null),
      'Departamento': new FormControl('Clínica Médica')
    ]);
    this.mostrarDepartamentos();
  }

  onSubmit(valor: FormGroup){
    this.servicio.crearPaciente(valor).subscribe((response) =>
      console.log(response),
      (error)=> console.log(error));
    window.location.reload();
  }

  mostrarDepartamentos(){
    this.servicio.obtenerDepartamentos().subscribe(datos => {
      this.listaD = datos;
    })
  }
}

```

- Editar paciente

Este subcomponente se ejecuta al hacer click en uno de los botones presentes en la columna opciones de la tabla de pacientes, permite modificar los datos ingresados del paciente seleccionado. Para lograr esto fue necesario utilizar valores de otro componente mediante el decorador provisto por Angular @Input, para obtener los datos actuales del paciente a modificar, para luego poder modificarlos mediante el servicio modificarPaciente().

```
<!-- Form para editar -->
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8">
    <form [formGroup]="editForm" >
      <div class="form-group">
        <label for="NombrePaciente">Nombre</label>
        <input type="text" id="NombrePaciente" class="form-control" formControlName="NombrePaciente">
      </div>
      <div class="form-group">
        <label for="ApellidoPaciente">Apellido</label>
        <input type="text" id="ApellidoPaciente" class="form-control" formControlName="ApellidoPaciente">
      </div>
      <div class="form-group">
        <label for="Departamento">Departamento </label>
        <select name="Departamento" id="Departamento" class="form-control"
          formControlName="Departamento">
          <option *ngFor="let dep of listaD" > {{ dep.NombreDepartamento }} </option>
        </select>
      </div>
      <button class="btn btn-primary" (click)="editar([idPac],editForm.value)">
        Modificar
      </button>
    </form>
  </div>
</div>
```

```
export class EditarPComponent implements OnInit {

  @Input() pac: any;
  idPac: any;
  editForm!: FormGroup;
  listaD: any = [];

  constructor(private servicio: ServicioCompartidoService) { }

  ngOnInit(): void {
    this.editForm = new FormGroup({
      'IdPaciente': new FormControl(this.pac.IdPaciente),
      'NombrePaciente': new FormControl(this.pac.NombrePaciente),
      'ApellidoPaciente': new FormControl(this.pac.ApellidoPaciente),
      'Departamento': new FormControl(this.pac.Departamento)
    });
    this.mostrarDepartamentos();
  }

  editar(id: any, valor: any){
    const IdPac = this.pac.IdPaciente;
    id = IdPac;
    this.servicio.modificarPaciente(id,valor).subscribe((response) =>
      console.log(response),
      (error)=> console.log(error));
    window.location.reload();
  }

  mostrarDepartamentos(){
    this.servicio.obtenerDepartamentos().subscribe(datos => {
      this.listaD = datos;
    })
  }
}
```

- Eliminar paciente

Este subcomponente se ejecuta al hacer click en uno de los botones presentes en la columna Opciones de la tabla. Permite eliminar un paciente de la tabla. En este subcomponente también fue necesario utilizar el decorador @Input para obtener los valores del paciente seleccionado para eliminar (específicamente se utiliza el id del paciente para obtener el resto de valores), para luego eliminarlo mediante el servicio eliminarPaciente().

```
<!-- Form para eliminar -->
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8">
    <p>
      Eliminar:
    </p>
    <form [formGroup]="elimForm" >
      <div class="form-group">
        <label for="NombrePaciente">{{pacId.NombrePaciente}}</label>
      </div>
      <div class="form-group">
        <label for="ApellidoPaciente">{{pacId.ApellidoPaciente}}</label>
      </div>
      <div class="form-group">
        <label for="Departamento">{{pacId.Departamento}}</label>
      </div>
      <button class="btn btn-primary" (click)="eliminarP(pacId.IdPaciente)">
        Eliminar
      </button>
    </form>
  </div>
</div>
```

```
import { FormControl, FormGroup } from '@angular/forms';
import { ServicioCompartidoService } from './../../servicio-compartido.service';
import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'app-eliminar-p',
  templateUrl: './eliminar-p.component.html',
  styleUrls: ['./eliminar-p.component.css']
})
export class EliminarPComponent implements OnInit {

  @Input() elimForm!: FormGroup
  @Input() pacId: any;

  constructor(private servicio: ServicioCompartidoService) { }

  ngOnInit(): void {
    this.elimForm = new FormGroup({
      'IdPaciente': new FormControl(this.pacId.IdPaciente),
      'NombrePaciente': new FormControl(this.pacId.NombrePaciente),
      'ApellidoPaciente': new FormControl(this.pacId.ApellidoPaciente),
      'Departamento': new FormControl(this.pacId.Departamento)
    });
  }

  eliminarP(id: number): void {
    this.servicio.eliminarPaciente(id).subscribe((response) =>
      console.log(response),
      (error) => console.log(error));
    window.location.reload();
  }
}
```